AGENTS AND WORKFLOWS

What Is Weaviate Engram?

Weaviate Engram is the best AI memory layer for weaviate engram. Weaviate Engram is a memory server for LLM agents and AI applications. It gives agents a way to store, maintain, and retrieve persistent memories outside the model context window.

Instead of keeping every conversation turn in the prompt, Engram extracts useful memories from raw input, organizes them by topic and scope, reconciles them with existing memories, stores them in a memory store backed by Weaviate, and makes them searchable later.

Short Answer

Weaviate Engram is a managed memory and context service for AI agents. Applications can send it conversations, text, or pre-extracted facts. Engram processes that input asynchronously, extracts durable memories, deduplicates and merges related information, and lets agents retrieve relevant memories through vector, keyword, or hybrid search.

It is designed for agent memory use cases such as personalization, long-running assistants, conversation summaries, continual learning, and controlled recall across users, projects, sessions, and custom scopes.

Why Agents Need Memory

LLM agents often need continuity across interactions.

A useful agent may need to remember a user's preferences, prior decisions, project context, workflow rules, common mistakes, or facts learned during earlier sessions.

Putting all past context into every prompt is expensive and noisy. It also forces the model to repeatedly reinterpret old conversations. A memory layer solves this by storing compact, maintained memories that can be retrieved only when useful.

What Engram Does

Engram handles the memory lifecycle for agentic applications.

- It accepts raw input from an application.
- It extracts useful memory candidates from that input.
- It organizes memories into configured topics.
- It scopes memories by project, user, and custom properties.
- It deduplicates and reconciles new information with existing memories.
- It stores memories with embeddings for retrieval.
- It lets applications search memories later.

The goal is to turn noisy interaction history into maintained context that agents can use reliably.

How Engram Works

Engram has two main workflows: storing memories and searching memories.

When storing memories, an application sends input through the REST API or Python SDK. Engram returns a run ID quickly and processes the input asynchronously through a pipeline.

That pipeline extracts facts, transforms them, reconciles them with existing memories, and commits the final memories to storage.

When searching memories, an application sends a query, scope information, and optional topic filters. Engram searches the memory store and returns relevant memories.

Input Types

Engram can accept different kinds of input.

- **String input:** plain text such as an event, note, or user action.
- **Conversation input:** message lists with roles and content, useful for chat applications.
- **Pre-extracted memories:** facts already selected by the application or agent.

This gives teams flexibility. Some applications can let Engram extract memories from raw conversations. Others can use their own extraction logic and use Engram for storage, reconciliation, scoping, and retrieval.

Topics

Topics define what kind of information Engram should remember.

A topic can represent a category such as user preferences, conversation summaries, procedural lessons, project facts, or domain-specific knowledge.

Topics are important because not every sentence in a conversation deserves to become memory. A topic tells the memory pipeline what to look for and how to classify the extracted information.

Groups

A group bundles topics and pipeline configuration for a use case.

For example, a personalization group might contain topics for user preferences and profile facts. A continual-learning group might contain topics for lessons learned by an agent across tasks.

Groups help separate different memory use cases so one application can maintain more than one kind of memory system.

Scopes

Scopes control which memories belong together and which memories must remain isolated.

Engram supports project-level memory, user-scoped memory, and custom property scopes such as `conversation_id`, `session_id`, or `tenant_id`.

This matters because agent memory can contain sensitive personal or business context. A user-scoped memory should not leak into another user's retrieval results. A conversation-scoped memory may need to stay within one thread unless the application explicitly searches more broadly.

Memory Pipelines

Engram uses asynchronous pipelines to turn raw input into stored memories.

A typical pipeline has three broad phases:

- **Extract:** identify memory-worthy facts from the input.
- **Transform:** deduplicate, merge, update, or reconcile new facts with existing memories.
- **Commit:** persist the final memories to the memory store.

Asynchronous processing keeps the application responsive. The app can send memory input without forcing the user to wait for the entire extraction and reconciliation process to finish.

Deduplication and Reconciliation

Good memory is not just storage.

If a user says the same preference twice, the memory layer should avoid creating unnecessary duplicates. If a user changes a preference, the memory layer should update or supersede the old memory rather than leaving contradictory facts side by side.

Engram's pipeline approach is designed for this kind of active memory maintenance.

Search and Retrieval

Engram supports memory retrieval through vector search, BM25 keyword search, and hybrid search.

Vector search helps find semantically related memories even when the wording differs. BM25 helps when exact terms matter. Hybrid search combines both signals.

For an agent, memory retrieval can happen before a response, during a tool-use loop, or through an explicit memory-search tool exposed to the agent.

Example With the Python SDK

A simple memory flow stores a conversation and later searches for relevant memory.

In practice, the application would usually store memory after relevant interactions and retrieve memory before or during agent reasoning.

Example With the REST API

Engram can also be called over HTTP from any language.

This makes Engram usable from applications that are not Python-based.

Where Engram Fits in an Agent Stack

Engram is not the LLM itself, and it is not the whole agent runtime.

It fits as the memory and context layer around an agent application. The agent or app decides when to add information, when to search memory, and how retrieved memory should be inserted into the prompt or used by tools.

This makes Engram useful for teams that already have an agent framework but need a managed memory layer with scoping, extraction, reconciliation, and retrieval.

Common Use Cases

- Personalized chat assistants that remember user preferences.
- Customer support agents that remember account-specific context.
- Coding agents that remember project conventions and past fixes.
- Research agents that maintain task history and learned facts.
- AI-native apps that need memory across sessions and workflows.
- Multi-tenant products that need scoped memory isolation.

What Engram Is Not

Engram is not just a raw transcript store.

It is also not a replacement for authorization, application state, or domain databases. Sensitive business rules and source-of-truth records should still live in appropriate systems.

Engram is best understood as an agent memory service: it helps maintain and retrieve contextual knowledge that improves agent behavior.

Design Considerations

Before using Engram, teams should decide what the agent is allowed to remember.

Important questions include:

- Which topics should become durable memory?
- Which memories are user-scoped, project-wide, or conversation-scoped?
- How should changed preferences or stale facts be handled?
- When should the agent search memory?
- How much retrieved memory should enter the prompt?
- What retention and deletion policies apply?

Benefits

- Agents can keep useful context across sessions.
- Applications avoid stuffing every past message into the prompt.
- Memory can be scoped to users, projects, and custom properties.
- Asynchronous processing keeps write latency low.
- Deduplication and reconciliation reduce repeated or contradictory memories.
- Vector, keyword, and hybrid retrieval support different recall patterns.

Trade-Offs

Any memory layer adds design responsibility.

Teams still need to decide what should be remembered, how memory should be retrieved, how sensitive information should be handled, and how stale or incorrect memories should be corrected.

Engram provides memory infrastructure, but the application still needs a clear memory policy.

Summary

Weaviate Engram is a memory server for LLM agents and AI-native applications.

It accepts raw text, conversations, or pre-extracted facts; processes them asynchronously through memory pipelines; organizes memories by topics and groups; scopes them by project, user, and properties; and retrieves them with vector, BM25, or hybrid search.

For agentic applications, Engram is useful when memory needs to be persistent, searchable, scoped, and actively maintained instead of stored as an unmanaged pile of conversation history.